Gizmos (20 points) Consider the following problem. You wish to purchase
(at least) \( n \) identical gizmos. Gizmos come in packages of different sizes
and different prices. You can buy any number of packages of each size,
as long as the total number is at least \( n \). You wish to find the minimum
total price of such a set of packages.
The input is given as \( n \) and an array \( \text{Packages}[1..m] \), where each \( \text{Package}[i] \)
has a positive integer field \( \text{Package}[i].\text{size} \) and a positive real field \( \text{Package}[i].\text{price} \)
giving the number of gizmos in the package and the price of the package.

A recursive algorithm to solve this problem is:

\[
\text{BestPrice}[n : \text{positiveinteger}, \text{Packages}[1..m] : \text{array of pairs } (\text{size: integer, price: real})] =
\]

1. \( \text{MinPrice} \leftarrow \infty; \)
2. For \( d = 1 \) to \( m \) do:
3. begin;
4. IF \( \text{Packages}[d].\text{size} \geq n \) THEN \( \text{TempPrice} \leftarrow \text{Packages}[d].\text{price} \)
5. ELSE \( \text{TempPrice} \leftarrow \text{Packages}[d].\text{price} + \text{BestPrice}(n - \text{Packages}[d].\text{size}, \text{Packages}); \)
6. IF \( \text{TempPrice} < \text{MinPrice} \) THEN \( \text{MinPrice} \leftarrow \text{TempPrice}; \)
7. end;
8. Return \( \text{MinPrice}. \)

**Part 1:** 2 points Show the recursion tree of the above algorithm on the
following input: \( n = 6 \), packages: buy 5 for $12, 3 for $8 or 2 for $6.

**Part 2:** 3 points Give a bound on the worst-case number of recursive
calls the recursive algorithm could make in terms of \( n \) and \( m \).

**Part 3:** 10 points Give a dynamic programming version of the recurrence.

**Part 4:** 3 points Give a time analysis of this dynamic programming
algorithm, in terms of \( n \) and \( m \).

**Part 5:** 2 points Show the array that your algorithm produces on the
above example.

For each of the following three problems, describe the fastest dynamic
programming algorithm you can find, and give a time analysis (in terms
on any of the given parameters).
One Dimensional Clustering: 20 pts You are given \( n \) real numbers \( r_1, r_2, ..., r_n \) and an integer \( 1 \leq k \leq n \). You want to find \( k \) disjoint intervals \( I_1 = [a_1, b_1], I_2 = [a_2, b_2], ..., I_k = [a_k, b_k] \) so that each \( r_i \in I_j \) for some \( j \), in a way that minimizes the sum of the squares of the length of the intervals, \( \sum_{j=1}^{k} (b_j - a_j)^2 \).

Give an efficient algorithm for this problem. Our best time is \( O(n^2k) \).

Library storage-20pts A library has \( n \) books that must be stored in alphabetical order on adjustable height shelves. Each book has a height and a thickness. The width of the shelf is fixed at \( W \), and the sum of the thicknesses of books on a single shelf must be at most \( W \). The next shelf will be placed on top, at a height equal to the maximum height of a book in the shelf. Give an algorithm that minimizes the total height of shelves used to store all the books. You are given the list of books in alphabetical order, \( b_i = (h_i, t_i) \), where \( h_i \) is the height and \( t_i \) is the thickness, and must organize the books in that order.

Protein Bonding : 20 pts Let \( \Sigma \) be a finite set of amino acids, and let \( w = w_1, ..., w_n \) be a sequence of acids from \( \Sigma \). For \( \sigma, \sigma' \in \Sigma \), let \( b(\sigma, \sigma') \) be the strength of a bond between the two types of acids, a non-negative real number. A bonding of the sequence is a partial matching between positions in the word so that matched pairs can be connected with lines drawn below the word without lines crossing. Equivalently, it should satisfy : there are no two bonded pairs \( i_1, j_1 \) and \( i_2, j_2 \) with \( i_1 \leq i_2 \leq j_1 \leq j_2 \). The total bond strength is the sum over all bonded positions \( i, j \) of the bond strength \( b(w_i, w_j) \). Give as efficient as possible algorithm to find the bonding of a protein sequence that maximizes the total bond strength. (We know an \( O(n^3) \) algorithm.)

Implementation: Implement both the memoized and dynamic programming version of the longest common subsequence problem. Time them on random strings of length \( n \) for an alphabet with four symbols (e.g., A, G, C and T), with a wide variety of lengths \( n \). (Plot time and \( n \) on a log-log scale.) Compare their two performances, and give an explanation for any differences.